DOCUMENT RESUME

ED 168 568 IR 007 109

AUTHOR Bearley, William L.
TITLE Non-Procedural Languages for Information Resource Management.
PUB DATE 30 Jan 79
NOTE 19p.; Paper presented at the International Meeting or Exchange (January 30, 1979)

EDRS PRICE MF01/PC01 Plus Postage.
DESCRIPTORS *Data Bases; *Data Processing; Information Retrieval; *Language Processing; *Management Systems; *Programing; Programing Languages; Programing Problems

ABSTRACT

The future of information resources management requires new approaches to implementing systems which will include a type of data base management that frees users to solve data processing problems logically by telling the system what they want, together with powerful non-procedural languages that will permit communication in simple, concise statements without complex data descriptions or program logic. A communication gap between the end-user and computer caused primarily by inadequacies of traditional programming approaches and methods of managing files may be remedied by the use of such non-procedural languages. The non-procedural language-base approach provides a "user-friendly" interface between user and computer that does not require knowledge of a data processing language or of data base structure. Decreased cost, increased productivity, and increased system availability to users are significant implications. The Information Resource Management System (IRMS), a non-procedural approach, is applicable for all internal organizational information processing, as a powerful development language for new applications, and as a utility for handing ad hoc requests or for one-time applications. The system focuses on the data, which are accessed through a data base manager. Generalized processing functions are accessed through a command language useable in both a batch and time-sharing environment, and a procedural language interface accommodates users with special processing needs. (CWM)

****
* Reproductions supplied by EDRS are the best that can be made *
* from the original document. *
****

ED168568

U.S. DEPARTMENT OF HEALTH,
EDUCATION & WELFARE
NATIONAL INSTITUTE OF
EDUCATION

THIS DOCUMENT HAS BEEN REPRODUCED EXACTLY AS RECEIVED FROM THE PERSON OR ORGANIZATION ORIGINATING IT. POINTS OF VIEW OR OPINIONS STATED DO NOT NECESSARILY REPRESENT OFFICIAL NATIONAL INSTITUTE OF EDUCATION POSITION OR POLICY

NON-PROCEDURAL LANGUAGES FOR INFORMATION RESOURCE MANAGEMENT

by

William L. Bearley

"PERMISSION TO REPRODUCE THIS MATERIAL HAS BEEN GRANTED BY

William L. Bearley

TO THE EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC) AND USERS OF THE ERIC SYSTEM."

IR007109

NON-PROCEDURAL LANGUAGES FOR INFORMATION RESOURCE MANAGEMENT

by

William L. Bearley
Copyright 1978

If we trace the development of programming languages over the past thirty years we find significant changes in the quantity of work done by the machine rather than by a programmer. Machine language required the programmer to precisely define each instruction in sequence in a code understood by only the machine and a few technicians. Symbolic languages allowed the use of symbols more familiar to people rather than machine code. Assemblers gave us macro capabilities along with subroutines for input, output and other common processes. However, we were still bound to the language of the machine for our instructions.

High level procedural languages such as FORTRAN and COBOL which were first implemented in the late 1950's provide communication to the machine on a level oriented to human problem solving rather than machine logic. However, we were still bound to defining exactly how to do a task and in what sequence instructions were to be executed.

At this point it is worthwhile asking "Why did programming become necessary?" The answer can be found in the communication gap between the end-user and the computer. Programming has been the only viable technique for this interaction between user and machine to take place. This gap is caused primarily by the inadequacies of traditional

programming approaches and by traditional methods of managing files.

The need to "program" in its traditional sense represents a major obstacle in information processing since most users, especially upper management, need timely, accurate data for decision making. Non-procedural languages can provide a "user-friendly" interface between the user and the computer that does not require knowledge of a data processing language or of the data base structure.

This concept of a non-procedural language is not new to the industry. Early examples are report generators and file managers which allow users to fill in the blanks on specification sheets. These are submitted to processors which contain the necessary logic to carry out the requests. Data base query languages are examples of an end-user interface to a data base. These languages allow users to make simple requests in terms of what they want. The system then provides the logic to carry out the request.

There are now powerful and flexible non-procedural languages available which provide enough capabilities for many installations to carry out all their data processing needs at a fraction of the overhead of traditional programming techniques. These languages are combined with data base management systems to provide complete data integration and independence from a user's point of view.

Rationale for use

While there are a number of reasons for using the non-procedural language-data base approach, there are three significant implications: 1) decreased cost, 2) increased productivity, and 3) increased system availability to users.

If you look at what is happening to costs in most installations you quickly discover people costs are going up while hardware costs are going down. Figure 1 represents the movement over time of most installations. In the keynote address for the Triennial IFIP Congress 77, Dr. Richard Tanaka, President of IFIP, indicated that at the current rate of change by 1985 90 percent or more of the costs of DP will be spent on training and support of people rather than hardware.

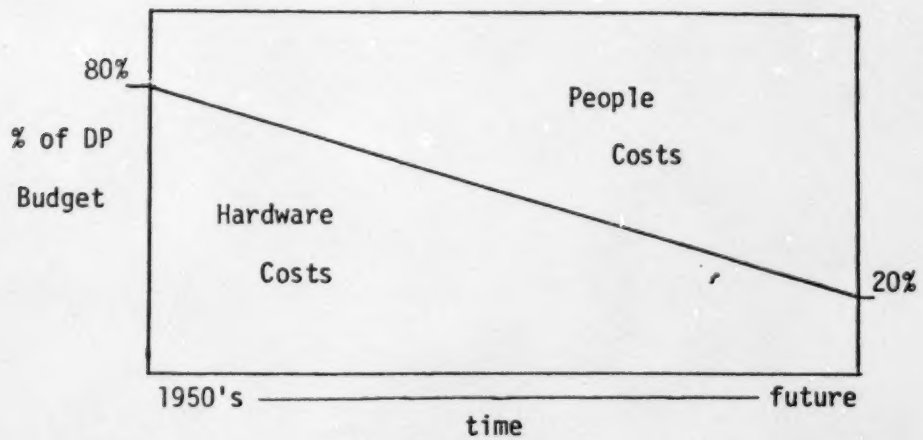

Figure 1

If we compare the development of programming languages in the same manner we find the amount of work required by a programmer has decreased while the amount of work done by the hardware has greatly increased with major strides being made at steps 4 and 6 as shown in Figure 2.

These advances in programming technology have been overcome by problems of maintenance caused by a backlog of operational applications developed using traditional languages. Maintenance can be divided into two categories: 1) Fixing errors in the program which

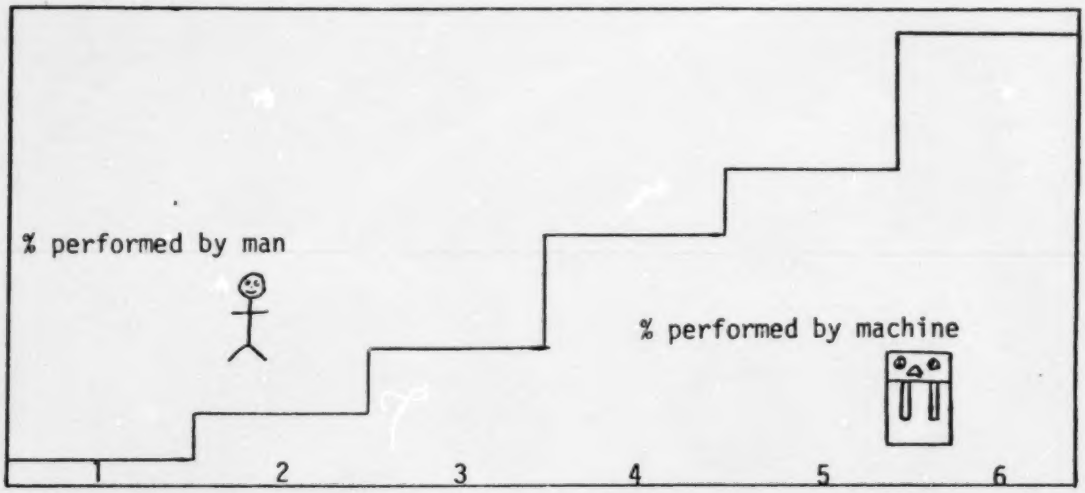

Figure 2

occur after it is in operation, 2) Adapting an application to a changing environment. Charles Lecht in a recent "In Depth" series in Computerworld estimated that on the average across the nation DP installations are spending between 60 percent and 70 percent of their programming time doing maintenance and that this will rise to 80 percent by 1985. Studies indicate the majority of this in a well-run installation is of the second type. Reports from installations using non-procedural languages show significant differences in this pattern as represented in Figure 3.

To understand this significant difference, compare the traditional programming approach with the non-procedural approach in preparing a program to print a simple report. Once the report is defined, the COBOL programmer must design the program logic, code the program,

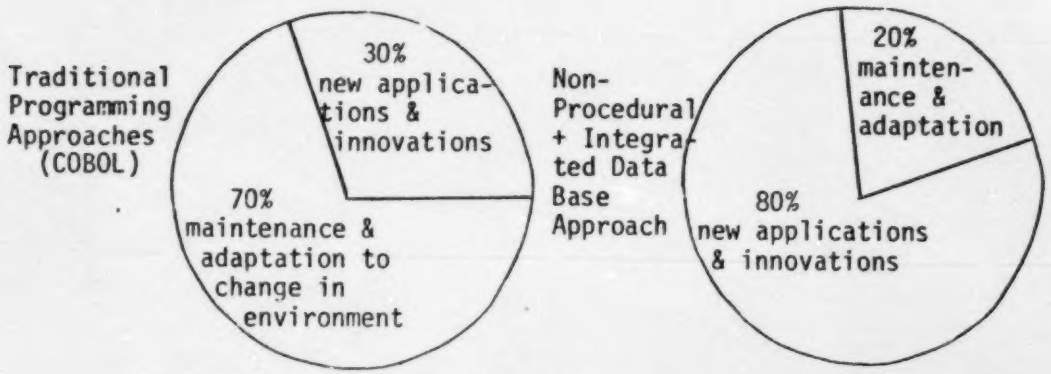
Traditional Programming Approaches (COBOL)
30% new applications & innovations
70% maintenance & adaptation to change in environment
Non-Procedural + Integrated Data Base Approach
20% maintenance & adaptation
80% new applications & innovations

Figure 3

debug the program, then finally run the program to get the results. It would not be unusual for a report with two levels of control break to take 200 to 300 lines. Using a non-procedural approach the report description can be directly defined to the system and may take around a dozen lines.

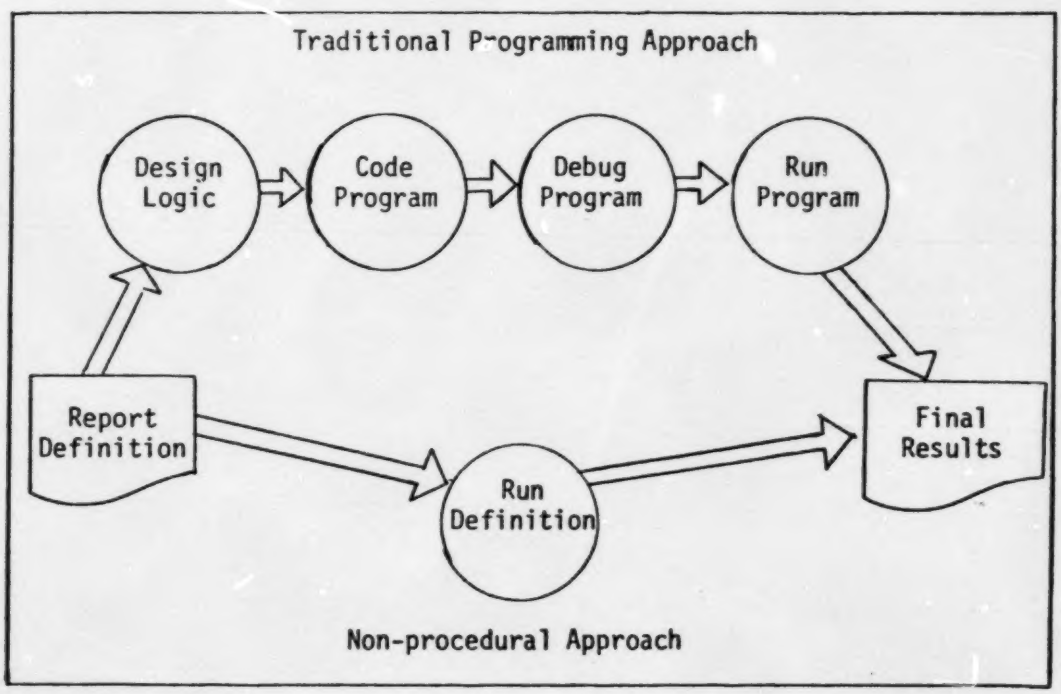

Figure 4

In terms of productivity a traditional programmer must design, code and debug the logic of how to do each report, then must maintain several hundred lines of code. With a non-procedural approach one need only define what they want and the system contains the logic of how to carry out the process. Therefore there are only a few lines of code which are easy to relate back to the report.

If you think of a typical traditional installation, how many times have they designed, coded and debugged file update logic, basic report logic, control break logic, record selection logic, etc.? How many lines of code does this "how to" logic take? What does it take to adapt all those programs to a changing environment? You can begin to understand the dilemma of many installations and why different approaches are necessary.

If you consider the typical interface between end-user and the data available in the data base, it is through the applications programmer or through a few fixed inquiry applications. If users have a new requirement they must wait while it is designed and programmed. Ad hoc demands are difficult, if not impossible, to meet in a timely manner. With the non-procedural approach ad hoc or new requests can often be done by the user. If not, the reaction time of programmers can be reduced from weeks or months to hours or days. The increase in system availability to users provided by such facilities goes a long way in answering complaints by users and management concerning the response time of DP.

Even more important in the user-DP interface is the ability to rapidly adapt to change. Given powerful and flexible languages

the adaptation to change becomes an easy process rather than the normal reactive, behind schedule, "we can't do that" approach.

Information Resource Management System (IRMS) - A non-procedural approach

To conceptually understand what a system for information management must contain, you must realize that it is a philosophy which frees the users (including programmers) to logically solve their data processing problems, telling the system what they want, not how to carry out the processing. It must be a complete system which incorporates data base with generalized data processing functions in a single package.

Such systems are often driven by data dictionaries which provide for processing and data independence. The data base must also be adaptable and relatively easy to change. Figure 5 depicts the components of the system. The central point is the data itself which is accessed through a data base manager. The generalized processing functions are accessed through a command language which can be used in both a batch and time-sharing environment. For users having special processing needs which cannot be accomodated by the functions provided, there is a procedural language interface.

One common complaint of users of such systems is that there are always some peculiar applications which do not lend themselves to non-procedural capabilities provided. To address this concern IRMS is hierarchical in nature providing more processing flexibility at each level. The most powerful level 3 provides procedural capabilities for special applications. The top level 1 provides the easiest

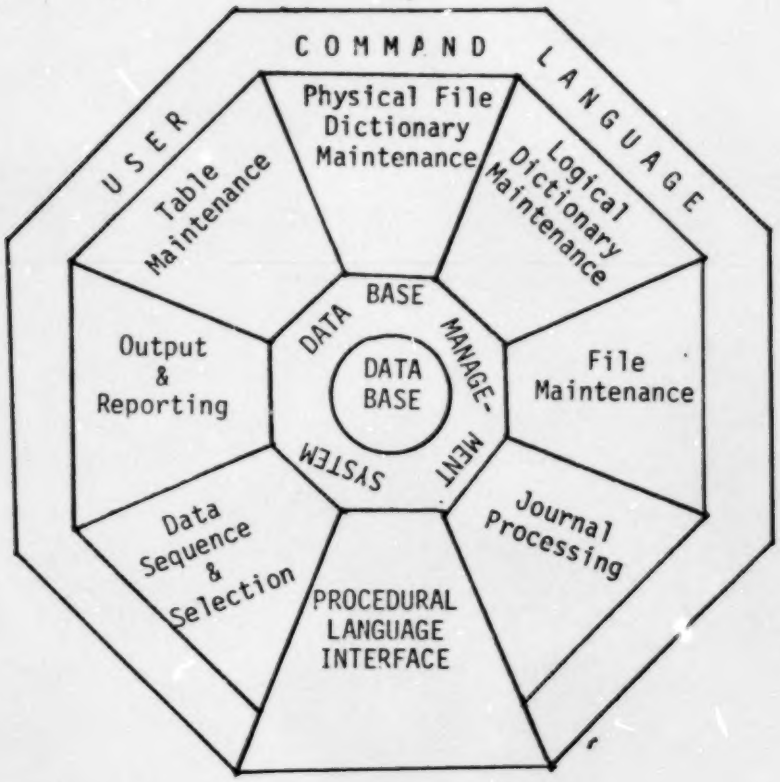

Figure 5

to use functions which are those used by people outside DP. To take full advantage of the system, a programmer would choose the highest level which will meet the requirements of the application.

Level 1 Non-procedural	Complete file management reporting and inquiry capabilities. Minimum of user interaction. Used for ad hoc reporting and for standard production requirements.
Level 2 Non-procedural	More user interaction required - more direct control. Used by programmers for special report formats and more complex file maintenance.
Level 3 Mix procedural & non-procedural	Used by programmers for complex applications and to add new non-procedural capabilities to the system. Provides complex data manipulation and I/O functions.

Level 3 can also be used to add new specialized, non-procedural functions to the system. One example of this was the addition of a simple text processing function to maintain the documentation for the system. Sophisticated on-line update/inquiry procedures are often done at this level.

Use

A system such as IRMS can be used for all information processing within an organization, as a powerful application development language for new applications, as a utility for handling ad hoc requests or for one-time applications. It can be used in a batch and on-line environment by programmers and users. It can carry out production applications as well as ad hoc requirements.

USE OF IRMS

Production (Programmers)	User
Routine batch and/or on-line file management and reporting	Clerk - pre-written on-line update
One-time (Q & D) applications	User - defined and maintained application
	User - ad hoc demands

The use of the system can be divided into two categories--use by DP and use by those outside DP. Within DP there are the routine day-to-day processing and the one-time demands. Routine processing is accomplished by programmers writing procedures at the appropriate level which are stored as files, then invoked by the system on demand. One-time demands can often be completed by operations using

the highest level of the non-procedural language. As long as the data is in the data base, the most complex one-time requests can be completed by a well-trained programmer in a matter of hours, if not minutes.

Access by users outside DP can be divided into three areas. Clerks will use procedures which are written by programmers. These are the typical on-line data collection, inquire or update applications. Users who have been trained to use the first level of non-procedural language will often write and store, for future use, procedures they find useful. Users can also carry out ad hoc requests in a matter of minutes to provide timely information for decision making.

An Example

On the following pages a sample report is shown along with a COBOL program which was written to print the report from a file. This program is over 125 statements. The statements required to produce exactly the same report using the non-procedural language are then shown.

To expand the example, assume you want the file sorted by city and alpha by name. Each city is to begin at the top of the page with a count of the number of students in each city. The additional COBOL code for the sort, control break and record count is not shown but would undoubtedly take another 20 to 30 lines in the DATA and PROCEDURE DIVISIONs. To accomplish this task using the non-procedural language you would need:

11/09/78 NAME & ADDRESS LISTING PAGE 1

ID NO	STUDENT NAME	STREET ADDRESS	CITY	ST	PHONE NO.
000017	BRISTER, SYLVIA J	123 FORTH ST	DUARTE	CA	213-098-3591
000043	ALTMAN, CAROL ANN	739 E PAMELA RD	COVINA	CA	213-096-3323
000072	ALISPONES, JOAN R	11478 W BADILLO	GLENDORA	CA	213-135-3356
000098	PORTER, PAUL ALLAN	1180 E CYPRESS	ONTARIO	CA	714-512-9837
000120	FLYNN, JOHN T	2809 SEA ISLAND TRL	DUARTE	CA	213-585-3589
000144	LITTLE, JOAN L	1407 RANDOM LANE	GLENDORA	CA	213-184-9632
000152	JAMES, BERNADINE A	759 E ADA	GLENDORA	CA	213-777-3355
000171	PARKS, RONALD R	841 E BENNETT	ARCADIA	CA	213-974-4464
000207	ROSS, ELIZABETH K	11430 FREER	MONROVIA	CA	213-632-3595
000218	SACKS, RICHARD	331 W LIME AVE	GLENDORA		213-040-3355
000221	WHITESCARVER, RHRD.	513 WHITCOMB AVE	GLENDORA	CA	213-788-3353
000225	CLARK, NANCY LINN	617 N BENDER AVE	GLENDORA	CA	213-800-9639
000231	NOKES, DERALD S	814 N WESTRIDGE	GLENDORA	CA	213-146-9638
000295	SIEGFRIED, JAMES M	6058 BRIARGATE LN	GLENDORA	CA	213-803-9635
000352	SIMPSON, LAURA ANN	1423 E DALTON AVE	MONROVIA	CA	213-959-3598
000362	TINKHAM, BARBARA	362 N MAYFLOWER AVE	GLENDORA	CA	213-426-3353
000420	ANDERSON, SHARON A	19532 RENWICK RD.	GLENDORA	CA	213-587-3355
000458	SANDELL, LYVIA M	447 N GLENDORA MT RD	GLENDORA	CA	213-544-9633
000471	GUARDALABENE, F R	868 E CYPRESS ST	GLENDORA	CA	213-203-3359
000548	OLSON JR, CHESTER A	339 N WASHINGTON	GLENDORA	CA	213-209-9630
000617	WHITE, PATRICK E	217 RODEO	MONROVIA	CA	213-496-3592
000714	GAUTREAU, CECILE A.	P O BOX 37	WEST COVINA	CA	759-3392
000750	CASTRUITA, MARY	1032 S. FLEETWELL	SAN DIMAS	CA	213-671-9666
000853	WETZEL, JANICE L	435 KIMBERLY	GLENDORA	CA	213-242-3352
000854	MELLINGER, WARREN L	134 WASHINGTON	MONROVIA	CA	213-426-3586
000905	CALDERWOOD, RICHARD	206 MELROSE	MONROVIA	CA	213-478-3594
000926	PONCE, RAY FRANCISCO	2567 S MYRTLE	GLENDORA	CA	213-205-9639
000981	WENDEL, WILLIAM A	19009 DUELL ST	AZUSA	CA	213-255-3347
000988	LENZ, MAX W	210 VICEROY	MONROVIA	CA	213-390-3585
001095	HAWK, DOREEN L	620 RANCHITO RD	GLENDORA	CA	213-289-3355
001122	SHEARS, RUTH, E	541 E CYPRESS	GLENDORA	CA	213-285-3353
001168	MC MAHON, FELICITA	19367 LOCHLEVEN	GLENDORA	CA	213-651-3359
001306	WILLIS, ROBERT J	410 W LAUREL	GLENDORA		213-392-3351
001322	MC DOWELL, OLIVE E.	19033 E ALFORD	AZUSA	CA	213-574-3359
001417	CREQUE, CLYDE J	6125 CALERA AVE	GLENDORA	CA	213-325-9632
001462	VALENTIJN, GUNVOR	1700 S GLENDORA #40	GLENDORA	CA	384-9633
001470	KING, KATHRYN MARIE	628 E. CYPRESS AVE	W. COVINA,	CA	213-202-3383
001475	KAMINSKI, MICHAEL L	520 N. YALETON	AZUSA	CA	213-126-9698
001489	HERNANDEZ, MAXIMINO	507 E 11TH	GLENDORA	CA	
001526	BRITTAIN, TERESA A	626 PLYMOUTH ST	MONROVIA	CA	213-052-3587
001556	BARKER, HAZEL MAY	228 LINWOOD	COVINA	CA	496-3387
001644	OSTROM- AMELIA	16127 KINGSIDE DR	AZUSA	CA	213-271-3342
001701	COURNOYER, KAREN E	206 N DALTON AVE	COVINA	CA	714-814-5990

```
IDENTIFICATION DIVISION.
 PROGRAM-ID.
    NAME-LISTING.
 AUTHOR.
    JOHN PROGRAMMER.
 INSTALLATION.
    THE-TYPICAL.
 DATE-WRITTEN.
    11/11/78.
 DATE-COMPILED.
    9/19/77.
 REMARKS.
    A NORMAL COBOL PROGRAM.
ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
  SOURCE-COMPUTER.
    XEROX-560.
  OBJECT-COMPUTER.
    XEROX-560.
  SPECIAL-NAMES.
    '1' IS TOP-OF-PAGE.
 INPUT-OUTPUT SECTION.
  FILE-CONTROL.
    SELECT NAME-FILE
          ASSIGN TO DISC.
    SELECT PRINT-FILE
          ASSIGN TO PRINTER.
DATA DIVISION.
 FILE SECTION.
  FD  NAME-FILE
           LABEL RECORD IS OMITTED
           RECORD CONTAINS 78 CHARACTERS
           DATA RECORD IS NAME-RECORD.
  01  NAME-RECORD.
    02  ID-IN                PICTURE X(6).
    02  NAME-IN              PICTURE X(20).
    02  STREET-IN            PICTURE X(20).
    02  CITY-IN              PICTURE X(15).
    02  ZIP-IN               PICTURE X(5).
    02  STATE-IN             PICTURE X(2).
    02  PHONE-IN.
      03  AREA-IN            PICTURE X(3).
      03  EXCH-IN              PICTURE X(3).
      03  PHO-IN             PICTURE X(4).
  FD  PRINT-FILE
           LABEL RECORD IS OMITTED
           RECORD CONTAINS 133 CHARACTERS
         - DATA RECORD IS PRINT-REC.
  01  PRINT-REC.
    02  CC                   PICTURE X(1).
    02  PRINT-LINE           PICTURE X(132).
```

```
WORKING-STORAGE SECTION.
 01  WORK-AREAS.
   02  PAGE-NUMBER          PICTURE S9(3) COMP VALUE 0.
   02  LINE-COUNT           PICTURE S9(2) COMP VALUE 99.
     88  END-OF-PAGE        VALUE 62 THRU 99.
 01  PAGE-HEADING.
   02  FILLER               PICTURE X.
   02  DATE-OUT             PICTURE X(8) VALUE '11/11/78'.
   02  FILLER               PICTURE X(21) VALUE SPACES.
   02  FILLER               PICTURE X(22)
                            VALUE 'NAME & ADDRESS LISTING'.
   02  FILLER               PICTURE X(21) VALUE SPACES.
   02  FILLER               PICTURE X(5) VALUE 'PAGE '.
   02  PAGE-OUT             PICTURE ZZ9.
 01  COL-HEADING.
   02  FILLER               PICTURE X.
   02  FILLER               PICTURE X(19)
                            VALUE 'ID NO  STUDENT NAME'.
   02  FILLER               PICTURE X(9) VALUE SPACES.
   02  FILLER               PICTURE X(25)
                            VALUE 'STREET ADDRESS       CITY'.
   02  FILLER               PICTURE X(12) VALUE SPACES.
   02  FILLER               PICTURE X(14) VALUE 'ST   PHONE NO.'.
 01  DETAIL-LINE.
   02  FILLER               PICTURE X.
   02  ID-OUT               PICTURE X(6).
   02  FILLER               PICTURE X VALUE SPACE.
   02  NAME-OUT             PICTURE X(20).
   02  FILLER               PICTURE X VALUE SPACE.
   02  STREET-OUT           PICTURE X(20).
   02  FILLER               PICTURE X VALUE SPACE.
   02  CITY-OUT             PICTURE X(15).
   02  FILLER               PICTURE X VALUE SPACE.
   02  STATE-OUT            PICTURE X(2).
   02  FILLER               PICTURE X(3) VALUE SPACES.
   02  PHONE-OUT.
     03  AREA-OUT           PICTURE X(3).
     03  FILLER             PICTURE X VALUE '-'.
     03  EXCH-OUT           PICTURE X(3).
     03  FILLER             PICTURE X VALUE '-'.
     03  PHO-OUT            PICTURE X(4).
 01  END-OF-JOB-LINE.
   02  FILLER               PICTURE X(21)
                            VALUE ' END OF SAMPLE REPORT'.
```

```
PROCEDURE DIVISION.
    OPEN INPUT NAME-FILE
         OUTPUT PRINT-FILE.
  PRINT-REPORT.
    READ NAME-FILE AT END
          GO TO END-OF-JOB.
    MOVE ID-IN TO ID-OUT.
    MOVE NAME-IN TO NAME-OUT.
    MOVE STREET-IN TO STREET-OUT.
    MOVE CITY-IN TO CITY-OUT.
    MOVE STATE-IN TO STATE-OUT.
    MOVE AREA-IN TO AREA-OUT.
    MOVE EXCH-IN TO EXCH-OUT.
    MOVE PHO-IN TO PHO-OUT
    IF END-OF-PAGE
      ADD 1 TO PAGE-NUMBER
      MOVE PAGE-NUMBER TO PAGE-OUT
      WRITE PRINT-REC FROM PAGE-HEADING
            AFTER TOP-OF-PAGE
      WRITE PRINT-REC FROM COL-HEADING
            AFTER ADVANCING 2 LINES
      MOVE SPACES TO PRINT-REC
      WRITE PRINT-REC
            AFTER ADVANCING 1 LINES
      MOVE 0 TO LINE-COUNT.
    WRITE PRINT-REC FROM DETAIL-LINE
          AFTER ADVANCING 1 LINES.
     COMPUTE LINE-COUNT = LINE-COUNT + 1.
    GO TO PRINT-REPORT.
  END-OF-JOB.
    WRITE PRINT-REC FROM END-OF-JOB-LINE
          AFTER ADVANCING 2 LINES.
    CLOSE NAME-FILE
          PRINT-FILE.
    STOP RUN.
```

EXAMPLE USING IRMS - A non-procedural approach

```
PRINT REPORT
HEADING 'NAME & ADDRESS LISTING'
FIELDS 1,2,3,4,5,3X,7
END
```

```
SORT FIELDS A4,A2
PRINT REPORT
HEADING 'NAME & ADDRESS LISTING BY CITY'
NEW PAGE 4
RECORD COUNT
FIELDS 1,2,3,4,5,3X,7
END
```

If this was a common request or a production procedure it could be stored and invoked by the system with a single command.

What does it mean

In my installation there were over 250 programs which provided the processing for the student records and the financial system. Estimating that the average size was around 500 lines (which may be low since many were in assembly language) this gives 125,000 lines of code to track and maintain. These systems were totally batch. The new system, which provides total on-line services plus batch reporting, requires 16,400 lines of procedure. It is totally integrated with a single data base.

Estimated cost of fully implementing a program runs between $5 and $7 per line.[1] Using these figures, 125,000 lines @ $5/line = $625,000; 16,400 lines @ $5/line = $82,000 or a difference of $543,000. Much of the cost of writing code in traditional languages is in the design of the logic and debugging the program once it is written. The cost of implementing one line of non-procedural code will be much less since there is little logic design time and debugging is much easier since there are considerably fewer statements. Assuming that the cost is closer to $2 per line, the savings is $593,000.

The time element is also significant. The previous student

system was written over a period of four years by three programmers. The new system was written in less than a month by one programmer.

Maintenance has been cut from being six months behind to adaptive maintenance of less than 10%. The way this is usually accomplished for reports is simply to discard the old report and while the user is describing the new report to the programmer they can be completing the specifications. Thus many new reports require a few minutes to implement.

We are now in a proactive mode of operation rather than a reactive mode. Adaptation due to change in the environment is handled as it is forseen and new applications are developed in a fraction of the time. Users carry out many of their own ad hoc requirements without contact with the computer center.

Users of other non-procedural systems report similar increases in productivity. A MARK IV user reports going from almost 100 percent of all staff time on maintenance to 90 percent on new development[2]. RAMIS users report a forty-to-one increase in programmer productivity over traditional methods[3]. A fully operational production control system for two manufacturing facilities was implemented in MIMS in five months by a small staff[4].

These non-procedural languages are natural extensions of data base. They provide a method for new installations to avoid the mistakes of the past and a way for existing installations to get out of the drudgery of maintenance into the world of innovation.

BIBLIOGRAPHY

1. Shaw, Donald R. "Selecting Software for the Business Mini." Computer Decisions, October 1978, pp. 58-59.

2. Flynn, Jim and Kimber, Dick. "From COBOL to MARK IV." Datamation, January 1977.

3. Mathematica Products Group. RAMIS II Rapid Access Management System. Princeton, N.J.: Mathematica Inc., 1977.

4. Driscoll, C. Thomas. "Programmerless DBMS Aids Manufacturer." Computerworld, November 21, 1977, p. 32.

5. Ziering, Charles. "High-Level Language: Revolution After DBMS. Computerworld, October 30, 1978.

6. Roach, Robert L. "Data Base/Practical Issues, Part 3 - High-Level Options Simplify Application Logic." Computerworld, September 1978.